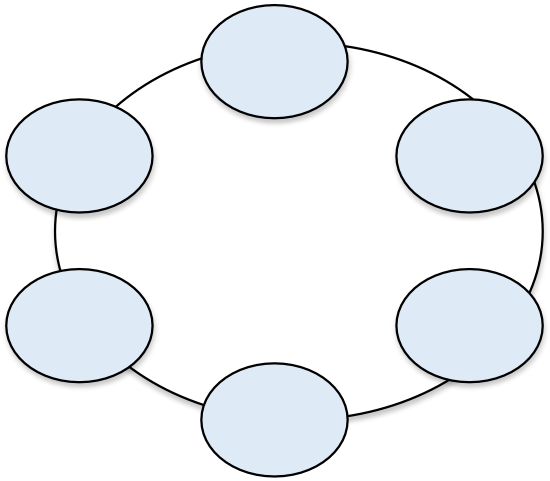
Apache Cassandra – An Introduction

- Apache Cassandra was born at Facebook. After Facebook open sourced the code in 2008, Cassandra became an Apache Incubator project in 2009 and subsequently became a top-level Apache project in 2010.

- It is a column-oriented database designed to support peer-to-peer symmetric nodes instead of the master−slave architecture.
- It is built on Amazon's dynamo and Google's BigTable.
- It is highly scalable, high performance distributed database
- It is column oriented database designed to support peer to peer network
- Adherance to availability and partition tolerance of CAP

Features:

1. Peer to Peer Network



-Designed to **distribute and manage large data loads** across multiple nodes in a **cluster constituted of commodity hardware**

-**Does NOT have a master slave architecture-NOT have a single point of failure**

-Graceful degradation **where everything does not** come **crashing at any instant owing a node failure**

- Ensures **data is distributed across all nodes** in the cluster

- **Each node exchange information across the cluster every second**

2. Gossip and Failure Detection

- Gossip protocol is **used for intra ring communication**
- Peer to peer communication protocol which **eases the discovery and sharing of location and state information with other nodes in the cluster**
- A node only **has to send out the communication to a subset of other nodes**

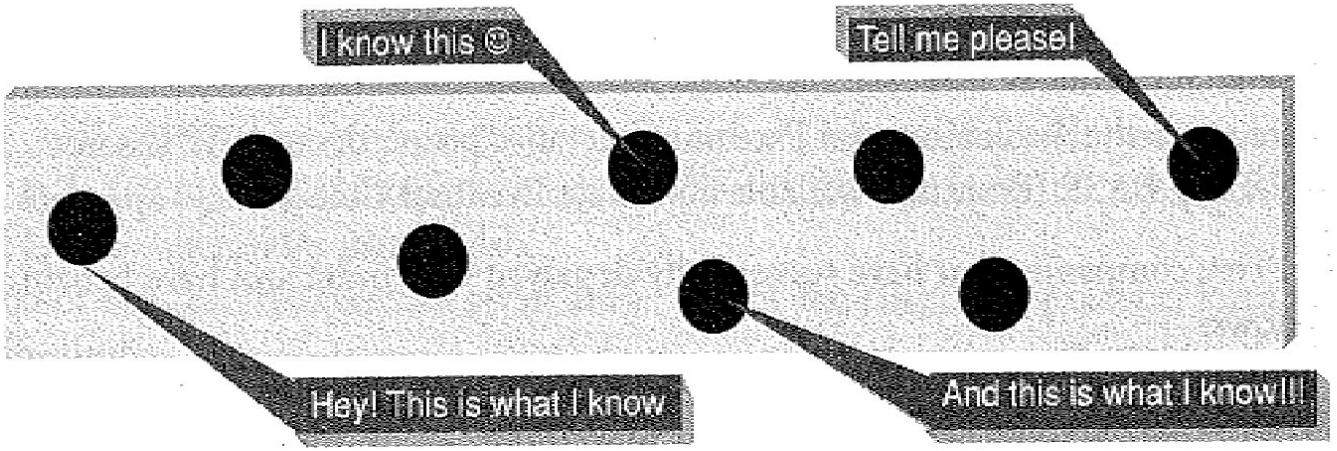


Figure 7.3 Gossip protocol.

3. Partitioner

- A partioner **takes a call on how to distribute data on the various nodes in the cluster**

- It also **determines the node on which to place the very first copy of the data**

- A **partitioner is a hash function is used to compute the token of the partion key**

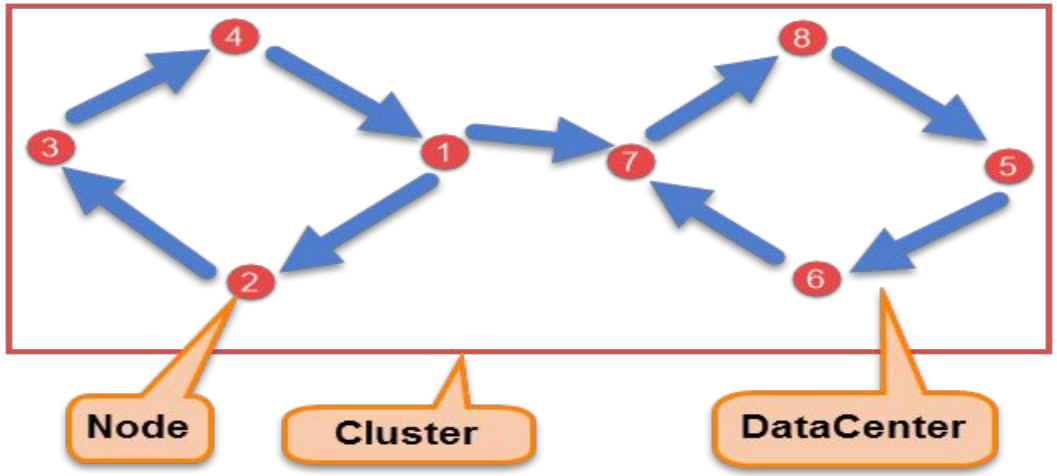
4. Replication factor

Node Node is the **place where data is sto**red. It is the basic component of Cassandra.

Data Center A **collection of nodes** are called data center. Many nodes are categorized as a data center.

rti

Cluster The cluster is the collection of many data centers.



As **hardware problem can occur** or **link can be down at any time during data process**, a **solution is required to provide a backup when the problem has occurred.**

So **data is replicated for assuring no single point of failure**.

Cassandra **places replicas of data on different nodes based on these two factors.**

1. Where to place next replica is determined by the **Replication Strategy.**

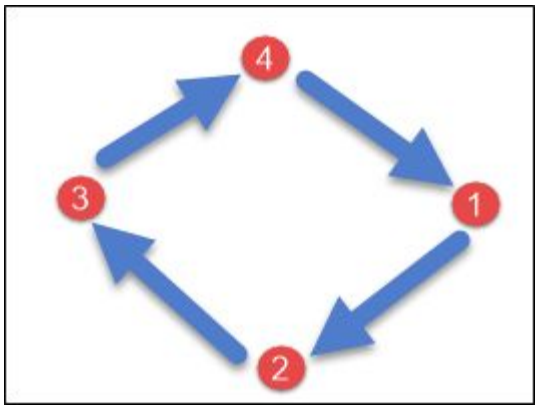
2.While the total number of replicas placed on different nodes is determined by the **Replication Factor**.

One Replication factor means that there is only a single copy of data while three replication factor means that there are three copies of the data on three different nodes.

There are two kinds of replication strategies in Cassandra.

1. SimpleStrategy

- SimpleStrategy is **used when you have just one data center**.
- SimpleStrategy **places the first replica on the node selected by the partitioner.**
- After that, **remaining replicas are placed in clockwise direction in the Node ring.**
- Here is the pictorial representation of the SimpleStrategy.



2. NetworkTopologyStrategy

- NetworkTopologyStrategy **is used when you have more than two data centers.**

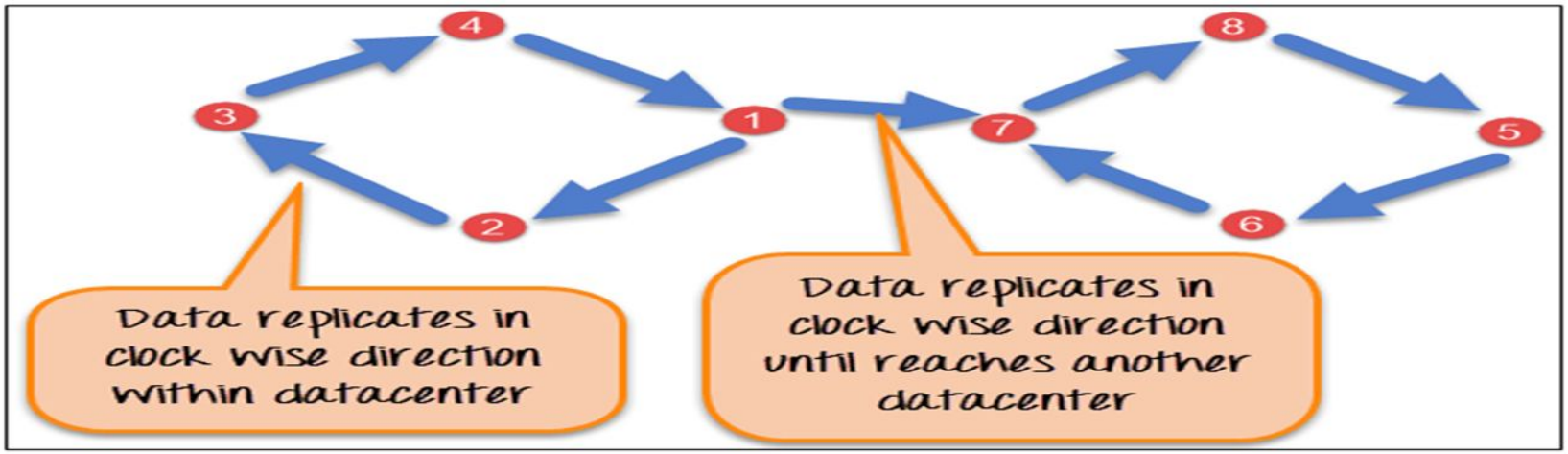
- In NetworkTopologyStrategy, **replicas are set for each data center separately.**

- NetworkTopologyStrategy places **replicas in the clockwise direction in the ring until reaches the first node in another rack**.

- This strategy **tries to place replicas on different racks in the same data center.**

- **This is due to the reason that sometimes failure or problem can occur in the rack**. Then replicas on other nodes can provide data.

- Here is the pictorial representation of the Network topology strategy



- **Read consistency means how many replicas must respond before sending out the result to client application**

- **Write consistency is how many replicas write must succeed before sending out an acknowledgement to the client application**

5. Anti Entropy and Read Repair

- A **client can connect to any node in the cluster to read data**

- **How many nodes will be read before responding to the client is based on the Consistency level specified by the client**

- If **few of the nodes respond with an out of date value** Cassandra will **initiate a read repair to bring the replicas with stale values up to date**

- This is **done using Anti Entropy gossip protocol**.

- Anti entropy implies **comparing all the replicas of each piece of data and updating each replica to the newest version**

6. Write operation in Cassandra

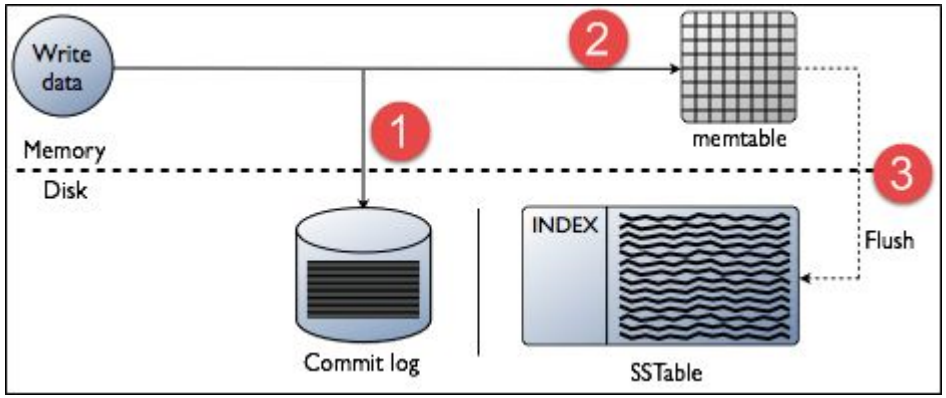
- When write request comes to the node, first of all, it logs in the commit log. Commit log is used for crash recovery.

  - After data written in Commit log, data is written in Mem-table

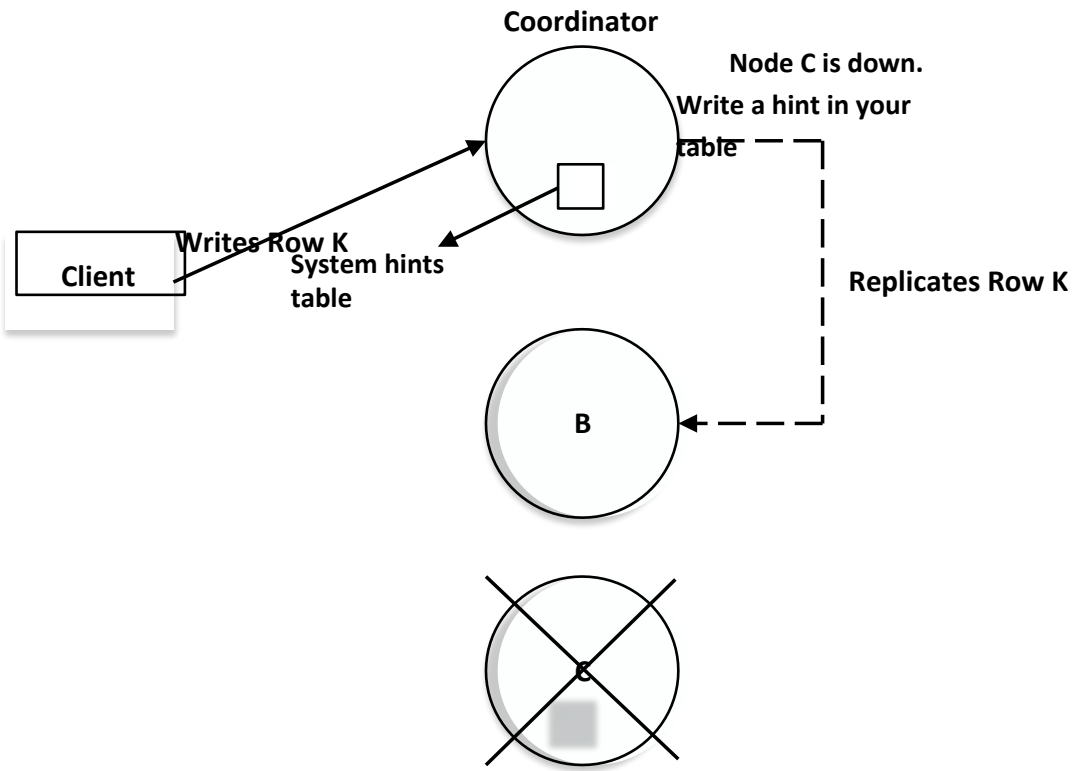
- Data written in the mem-table on each write request also writes in commit log separately.

- Mem-table is a temporarily stored data in the memory while Commit log logs the transaction records for back up purposes or crash recovery

- When mem-table is full, data is flushed to the SSTable data file.



7. Hinted Handoffs



Three nodes A,B, C

C is down

Replication factor is 2

Write operation on node A which is the coordinator and serves as a proxy

When row k is wriiten by the client to Node A, it will write row K to node B and stores a hint for Node C.

Hint has the following information

1. **Location of the node on which replica is to be placed**
2. **Version Mtada**
3. **Actual data**

8. Tunable Consistency

- Strong Consistency- Each update propagates to all location where that piece of data resides

- Eventual Consistency- Client is acknowledged with success as soon as part of the cluster acknowledges the write

Copyright 2015, WILEY INDIA PVT. LTD.

Read Consistency

ONE	**Returns a response from the closest node (replica) holding the data.**
QUORUM	Returns a result from a quorum of servers with the most recent timestamp for the data.
LOCAL_QUORUM	Returns a result from a quorum of servers with the most recent timestamp for the data in the same data center as the coordinator node.
EACH_QUORUM	Returns a result from a quorum of servers with the most recent timestamp in all data centers.
ALL	This provides the highest level of consistency of all levels and the lowest level of availability of all levels. It responds to a read request from a client after all the replica nodes have responded.

Copyright 2015, WILEY INDIA PVT. LTD.

Write Consistency

ALL	**This is the highest level of consistency of all levels as it necessitates that a write must be written to the commit log and Memtable on all replica nodes in the cluster.**
EACH_QUORUM	A write must be written to the commit log and Memtable on a quorum of replica nodes in all data centers.
QUORUM	A write must be written to the commit log and Memtable on a quorum of replica nodes.
LOCAL_QUORUM	A write must be written to the commit log and Memtable on a quorum of replica nodes in the same data center as the coordinator node. This is to avoid latency of inter-data center communication.
ONE	A write must be written to the commit log and Memtable of at least one replica node.
TWO	A write must be written to the commit log and Memtable of at least two replica nodes.
THREE	A write must be written to the commit log and Memtable of at least three replica nodes.
LOCAL_ONE	A write must be sent to, and successfully acknowledged by, at least one replica node in the local data center.

Copyright 2015, WILEY INDIA PVT. LTD.

Cassandra - Data Model

The data model of Cassandra is significantly different from what we normally see in an RDBMS.

Cluster
Cassandra database is distributed over several machines that operate together.

The outermost container is known as the Cluster.

For failure handling, every node contains a replica, and in case of a failure, the replica takes charge.

Cassandra arranges the nodes in a cluster, in a ring format, and assigns data to them.

Keyspace

Keyspace is the outermost container for data in Cassandra. The basic attributes of a Keyspace in Cassandra are −

Replication factor − It is the number of machines in the cluster that will receive copies of the same data.

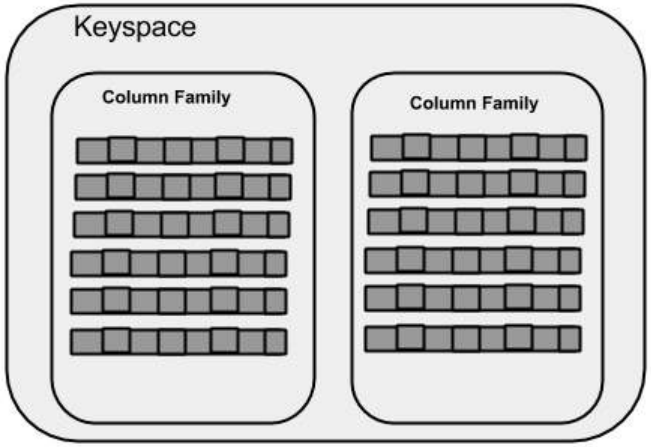
Replica placement strategy − It is nothing but the strategy to place replicas in the ring. We have strategies such as **simple strategy** (rack-aware strategy), **old network topology strategy** (rack-aware strategy), and **network topology strategy** (datacenter-shared strategy).

Column families − Keyspace is a container for a list of one or more column families.
A column family, in turn, is a container of a collection of rows.
Each row contains ordered columns.
Column families represent the structure of your data.
Each keyspace has at least one and often many column families.

The syntax of creating a Keyspace is as follows −

```
CREATE KEYSPACE Keyspace name WITH replication = {'class':
'SimpleStrategy', 'replication_factor' : 3};
```

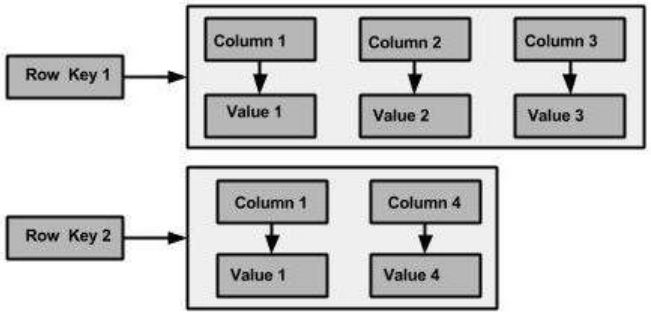
The following illustration shows a schematic view of a Keyspace.

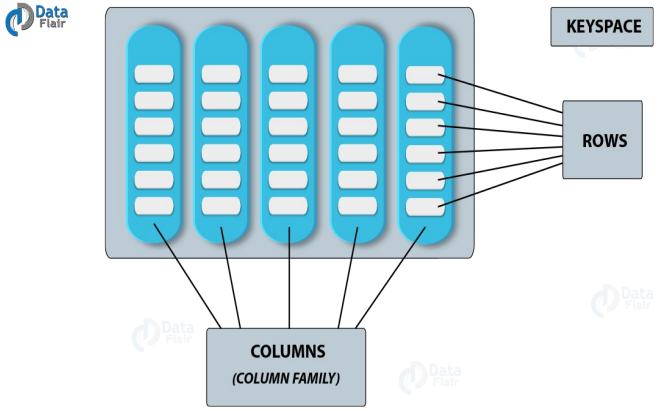


Column Family

A column family is a container for an ordered collection of rows. Each row, in turn, is an ordered collection of columns

The following figure shows an example of a Cassandra column family.





The following table lists down the points that differentiate the data model of Cassandra from that of an RDBMS.

RDBMS	Cassandra
RDBMS deals with structured data.	Cassandra deals with unstructured data.
It has a fixed schema.	Cassandra has a flexible schema.
In RDBMS, a table is an array of arrays. (ROW x COLUMN)	In Cassandra, a table is a list of “nested key-value pairs”. (ROW x COLUMN key x COLUMN value)
Database is the outermost container that contains data corresponding to an application.	Keyspace is the outermost container that contains data corresponding to an application.
Tables are the entities of a database.	Tables or column families are the entity of a keyspace.
Row is an individual record in RDBMS.	Row is a unit of replication in Cassandra.
Column represents the attributes of a relation.	Column is a unit of storage in Cassandra.
RDBMS supports the concepts of foreign keys, joins.	Relationships are represented using collections.

Rowid	Empid	Lastname	Firstname	Salary
001	10	Smith	Joe	60000
002	12	Jones	Mary	80000
003	11	Johnson	Cathy	94000
004	22	Jones	Bob	55000

10:001,12:002,11:003,22:004;
Smith:001,Jones:002,Johnson:003,Jones:004;
Joe:001,Mary:002,Cathy:003,Bob:004;
60000:001,80000:002,94000:003,55000:004;

```
CREATE KEYSPACE Students WITH REPLICATION = {
        'class':'SimpleStrategy',
        'replication_factor':1
};
```

Objective: To describe all the existing keyspaces.

Act:

```
DESCRIBE KEYSPACES;
```

Objective: To get more derails on the existing keyspaces such as keyspace name, durable writes, strategy class, strategy options, etc.

Act:

```
SELECT *
    FROM system.schema_keyspaces;
```

Objective: To use the keyspace "Students", use the following command:

Use keyspace_name

Use connects the client session to the specified keyspace.

Act:

USE Students;

Objective: To create a column family of the ...

Act:

```
CREATE TABLE Student_Info (
   RollNo int PRIMARY KEY,
   StudName text,
   DateofJoining timestamp,
   LastExamPercent double
   );
```

Outcome:

```
cqlsh> use Students;
cqlsh:students> CREATE TABLE Student_Info (
            ...    RollNo int PRIMARY Key,
            ...    StudName text,
            ...    Dateofjoining timestamp,
            ...    LastExamPercent double
            ...    );
```

... in the keyspace "students".

Explanation about the composite PRIMARY KEY:

Primary key (column_name1, *column_name2, column_name3* ...)

Primary key ((column_name4, *column_name5*), column_name6, *column_name7* ...)

In the above syntax,

column_name1 is the partition key

column_name2 and column_name3 are the clustering columns.

column_name4 and column_name5 are the partitioning keys

column_name6 and column_name7 are the clustering columns.

The partition key is used to distribute the data in the table across various nodes that constitute the cluster. The clustering columns are used to store data in ascending order on the disk.

Objective: To lookup the names of all tables in the current keyspace, or in all the keyspaces if there is no current keyspace.

Act:

DESCRIBE TABLES;

Outcome:

```
cqlsh:students> describe tables;

student_info
```

Objective: To describe the table "student_info" use the below command.

Act:

DESCRIBE TABLE student_info;

7.6 CRUD (CREATE, READ, UPDATE, AND DELETE) OPERATIONS

Objective: To insert data into the column family "student_info'

Act:

```
BEGIN BATCH
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (1,'Michael Storm','2012-03-29', 69.6)
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (2,'Stephen Fox','2013-02-27', 72.5)
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (3,'David Flemming','2014-04-12', 81.7)
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (4,'Ian String','2012-05-11', 73.4)
APPLY BATCH;
```

Objective: To view the data from the table "student_info".

Act:

```
SELECT *
        FROM student_info;
```

The above select statement retrieves data from the "student_info" table.

ve: To view only those records where the RollNo column either has a value 1 or 2 or 3.

CT *

FROM student_info

WHERE RollNo IN(1,2,3);

or the above statement to execute successfully, ensure that the following criteria are satisfied:

er the partition key definition includes the column that is used in the where clause i.e. search ria.

the column being used in the where clause, that is, search criteria, has an index defined on it g the CREATE INDEX statement.

ne:

```
udents> Select * from student_info where RollNo IN(1,2,3);
 dateofjoining
```

Let us try running a query with "studname" in the where clause. Since "studname" is neither the primary key column nor a column in the primary key definition and also does not have an index defined on it, such a query will lead to error.

We set the stage to resolve the error by creating an index on the "studname" column of the "student_info" table and then subsequently executing the query.

To create an index on the "studname" column of the "student_info" column family use

```
CREATE INDEX ON student_info(studname)
```

To execute the query using the index defined on "studname" column use

```
SELECT *
    FROM student_info
        WHERE studname='Stephen Fox' ;
```

Outcome:

```
sh:students> create index on student_info(studname);
sh:students> select * from student_info where studname='Stephen Fox' ;

rollno | dateofjoining                              | lastexampercent | studname
-------+--------------------------------------------+-----------------+-----------
     2 | 2013-02-27 00:00:00India Standard Time |            72.5 | Stephen Fox

1 rows)
```

Objective: Let us create another index on the "LastExamPercent" column of the "student_info" column family.

Act:

```
CREATE INDEX ON student_info(LastExamPercent);
```

Outcome:

```
cqlsh:students> create index on student_info(LastE
cqlsh:students> select * from student_info where LastExamPercent = 81.7;

 rollno | dateofjoining                          | lastexampercent | studname
--------+----------------------------------------+-----------------+----------------
      3 | 2014-04-12 00:00:00India Standard Time |            81.7 | David Flemming

(1 rows)
```

Objective: To specify the number of rows returned in the output using limit.

Act:

```
SELECT rollno, hobbies, language, lastexampercent
        FROM student_info LIMIT 2;
```

Outcome:

```
cqlsh:students> select rollno, hobbies, language, lastexampercent from student_info limit 2;

 rollno | hobbies                                | language          | lastexampercent
--------+----------------------------------------+-------------------+-----------------
      1 |                {'Chess, Table Tennis'} | ['Hindi, English'] |            69.6
      4 | {'Lawn Tennis, Table Tennis, Golf'} | ['Hindi, English'] |            73.4

(2 rows)
```

Act:

UPDATE student_info SET StudName = 'David Sheen' WHERE RollNo = 2;

Outcome:

```
cqlsh:students> UPDATE student_info  SET StudName = 'David Sheen'  WHERE RollNo = 2;
cqlsh:students> select * from student_info where rollno = 2;

 rollno | dateofjoining                              | lastexampercent | studname
--------+--------------------------------------------+-----------------+-------------
      2 | 2013-02-27 00:00:00India Standard Time |            72.5 | David Sheen

(1 rows)

cqlsh:students>
```

Objective: Let us try updating the value of a primary key column.

Act:

UPDATE student_info SET rollno=6 WHERE rollno=3;

Outcome:

```
cqlsh:students> update student_info set rollno=6 where rollno=3;
Request: PRIMARY KEY part rollno found in SET part
cqlsh:students>
```

Note: It does not allow update to a primary key column.

Objective: Updating more than one column of a row of Cassandra table.

Act:

Step 1: Before the update

```
cqlsh:students> select rollno, studname, lastexampercent from student_info where rollno=3;

 rollno | studname        | lastexampercent
--------+-----------------+-----------------
      3 | David Flemming  |            81.7

(1 rows)
```

Step 2: Applying the update

```
cqlsh:students> select rollno, studname, lastexampercent from student_info where rollno=3;

 rollno | studname | lastexampercent
--------+----------+-----------------
      3 |  Samaira |              85

(1 rows)
```

Step 3: After the update

```
cqlsh:students> DELETE LastExamPercent FROM student_info where RollNo=2;
cqlsh:students> select * from student_info where rollno = 2;

 rollno | dateofjoining                                | lastexampercent | studname
--------+----------------------------------------------+-----------------+-------------
      2 | 2013-02-27 00:00:00India Standard Time |            null | David Sheen

(1 rows)

cqlsh:students>
```

Objective: To delete the column "LastExamPercent" from the "student_info" table for the rec... where the RollNo = 2.

Note: Delete statement removes one or more columns from one or more rows of a Cassandra table ... removes entire rows if no columns are specified.

Act:

```
DELETE LastExamPercent FROM student_info WHERE RollNo=2;
```

Outcome:

```
cqlsh:students> DELETE LastExamPercent FROM student_info where RollNo=2;
cqlsh:students> select * from student_info where rollno = 2;

 rollno | dateofjoining                                | lastexampercent | studname
--------+----------------------------------------------+-----------------+-----------
      2 | 2013-02-27 00:00:00India Standard Time |            null | David Sheen

(1 rows)

cqlsh:students>
```

Objective: To delete a row (where RollNo = 2) from the table "student_info".

Act:

```
DELETE FROM student_info WHERE RollNo=2;
```

Outcome:

```
cqlsh:students> DELETE  FROM student_info where RollNo=2;
cqlsh:students> select * from student_info where rollno=2;

(0 rows)

cqlsh:students>
```

Objective: To create a table "project _details" with primary key as (project_id, project_name).

Act:

```
CREATE TABLE project_details (
        project_id int,
        project_name text,
        stud_name text,
        rating double,
        duration int,
        PRIMARY KEY (project_id, project_name));
```

Objective: To insert data into the column family "project_details".

```
BEGIN BATCH
INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
VALUES (1,'MS data migration','David Sheen',3.5,720)
INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
VALUES (1,'MS Data Warehouse','David Sheen',3.9,1440)
INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
VALUES (2,'SAP Reporting','Stephen Fox',4.2,3000)
INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
VALUES (2,'SAP BI DW','Stephen Fox',4,9000)
APPLY BATCH;
```

Objective: To view all the rows of the "project_details" table.

Query:

```
SELECT *
        FROM project_details;
```

Objective: To view row/record from the "project_details" table wherein the project_id=1.

Act:

```
SELECT *
        FROM project_details
                WHERE project_id=1;
```

Outcome:

```
cqlsh:students> Select * from project_details where project_id=1;

 project_id | project_name       | duration | rating | stud_name
------------+--------------------+----------+--------+------------
          1 | MS Data Warehouse  |     1440 |    3.9 | David Sheen
          1 | MS data Migration  |      720 |    3.5 | David Sheen

(2 rows)
```

Set

7.7.1 Set Collection

A column of type set consists of unordered unique values. However, when the column is queried, it returns the values in sorted order. For example, for text values, it sorts in alphabetical order.

7.7.2 List Collection

When the order of elements matter, one should go for a list collection. For example, when you store the preferences of places to visit by a user, you would like to respect his preferences and retrieve the values in the order in which he has entered rather than in sorted order. A list also allows one to store the same value multiple times.

7.7.3 Map Collection

As the name implies, a map is used to map one thing to another. A map is a pair of typed values. It is used to store timestamp related information. Each element of the map is stored as a Cassandra column. Each element can be individually queried, modified, and deleted.

Objective: To alter the schema for the table "student_info" to add a column "hobbies".

Act:

```
ALTER TABLE student_info ADD hobbies set<text>;
```

Objective: To update the table "student_info" to provide the values for "hobbies" for the student with Rollno =1.

Act:

```
UPDATE student_info
        SET hobbies = hobbies + {'Chess, Table Tennis'}
                WHERE RollNo=1;
```

Outcome:

```
cqlsh:students> UPDATE student_info
           ...    SET hobbies = hobbies + {'Chess, Table Tennis'} WHERE RollNo=1;
```

To confirm the values in the hobbies column, use the below command:

```
SELECT *
        FROM student_info
                WHERE RollNo=1;
```

```
cqlsh:students> select * from student_info where rollno=1;

 rollno | dateofjoining                          | hobbies                  | language | lastexampercent | studname
--------+----------------------------------------+--------------------------+----------+-----------------+--------------
      1 | 2012-05-29 00:00:00India Standard Time | ['Chess, Table Tennis'] |     null |            60.8 | Michael Storm

(1 rows)
```

Objective: To alter the schema of the table "student_info" to add a list column "language".

Act:

```
ALTER TABLE student_info ADD language list<text>;
```

Objective: To update values in the list column, "language" of the table "student_info".

Act:

```
UPDATE student_info
    SET language = language + ['Hindi, English']
        WHERE RollNo=1;
```

Outcome:

More Practice on Collections (SET and LIST)

```
CREATE TABLE users (
        user_id text PRIMARY KEY,
        first_name text,
        last_name text,
        emails set<text>
);
```

Objective: To insert values into the "emails" column of the "users" table.

Note: Set values must be unique.

Act:

```
INSERT INTO users
        (user_id, first_name, last_name, emails)
                VALUES('AB', 'Albert', 'Baggins', {'a@baggins.com', 'baggins@gmail.com'});
```

Objective: Add an element to a set using the UPDATE command and the addition (+) operator.

Act:

```
UPDATE users
    SET emails = emails + {'ab@friendsofmordor.org'}
        WHERE user_id = 'AB';
```

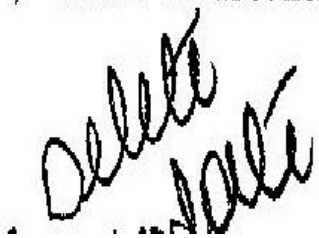
Objective: To retrieve email addresses for Albert from the set.

Act:

```
SELECT user_id, emails
        FROM users
                WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> SELECT user_id, emails FROM users WHERE user_id = 'AB';

 user_id | emails
---------+-----------------------------------------------------------------
      AB | {'a@baggins.com', 'ab@friendsofmordor.org', 'baggins@gmail.com'}

(1 rows)
```



Objective: To remove an element from a set using the subtraction (–) operator.

Act:

```
UPDATE users
    SET emails = emails - {'ab@friendsofmordor.org'}
        WHERE user_id = 'AB';
```

Objective: To remove all elements from a set by using the UPDATE or DELETE statement.

Act:

```
UPDATE users
    SET emails = {}
        WHERE user_id = 'AB';
```

```
cqlsh:students> UPDATE users SET emails = {} WHERE user_id = 'AB';
```

OR

```
DELETE emails
    FROM users
        WHERE user_id = 'AB';
```

Objective: To alter the "users" table to add a column, "top_places" of type list.

Act:

```
ALTER TABLE users ADD top_places list<text>;
```

Objective: To update the list column "top_places" in the "users" table for user_id = 'AB'.

Act:

```
UPDATE users
        SET top_places = [ 'Lonavla', 'Khandala' ]
                WHERE user_id = 'AB';
```

Objective: Prepend an element to the list by enclosing it in square brackets and using the addition (+) operator.

Act:

```
UPDATE users
    SET top_places = [ 'Mahabaleshwar' ] + top_places
        WHERE user_id = 'AB';
```

Objective: To append an element to the list by switching the order of the new element data and the list name in the update command.

Act:

```
UPDATE users
    SET top_places = top_places + [ 'Tapola' ]
        WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> select * from users;

 user_id | emails | first_name | last_name | top_places
---------+--------+------------+-----------+-----------------------------------------------------
      AB |   null |     Albert |   Baggins | ['Mahabaleshwar', 'Lonavla', 'Khandala', 'Tapola']
```

Objective: To query the database for a list of top places.

Act:

```
SELECT user_id, top_places
        FROM users
                WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> SELECT user_id, top_places FROM users WHERE user_id = 'AB';

 user_id | top_places
---------+-----------------------------------------------------------
      AB | ['Mahabaleshwar', 'Lonavla', 'Khandala', 'Tapola']

(1 rows)
```

Objective: To remove an element from a list using the DELETE command and the list index position in square brackets.

The record as it exists prior to deletion is

```
cqlsh:students> SELECT user_id, top_places FROM users WHERE user_id = 'AB';

 user_id | top_places
---------+-----------------------------------------------------------
      AB | ['Mahabaleshwar', 'Lonavla', 'Khandala', 'Tapola']

(1 rows)
```

Act:

```
DELETE top_places[3]
    FROM users
        WHERE user_id = 'AB';
```

```
cqlsh:students> DELETE top_places[3] FROM users WHERE user_id = 'AB';
```

Outcome: The status after deletion is

```
cqlsh:students> select * from users;

 user_id | emails | first_name | last_name | top_places
---------+--------+------------+-----------+-----------------------------------------
      AB |   null |     Albert |   Baggins | ['Mahabaleshwar', 'Lonavla', 'Khandala']

(1 rows)
```

7.7.5 Using Map: Key, Value Pair

Objective: To alter the "users" table to add a map column "todo".

Act:

```
ALTER TABLE users
        ADD todo map<timestamp, text>;
```

Objective: To update the record for user (user_id = 'AB') in the "users" table.

Act: The record from user_id = 'AB' as it exists in the "users" table is

```
cqlsh:students> select * from users where user_id='AB';

 user_id | emails | first_name | last_name | todo | top_places
---------+--------+------------+-----------+------+-----------------------------------------
      AB |   null |     Albert |   Baggins | null | ['Mahabaleshwar', 'Lonavla', 'Khandala'

(1 rows)
```

```
cqlsh:students> UPDATE users
            ...   SET todo =
            ...   {'2014-9-24':'Cassandra Session',
            ...    '2014-10-2 12:00' :'MongoDB Session'}
            ...   WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> select user_id, todo from users where user_id='AB';

r_id | todo
-----+-----------------------------------------------------------------------------------------
  AB | {'2014-09-24 00:00:00India Standard Time': 'Cassandra Session', '2014-10-02 12:00:00India Standard Time': 'MongoDB S
```

Objective: To delete an element from the map using the DELETE command and enclosing the timestamp of the element in square brackets.

Act:

```
DELETE todo['2014-9-24']
        FROM users
                WHERE user_id = 'AB';
```

```
user_id | todo
--------+-----------------------------------------------------------------
     AB | {'2014-10-02 12:00:00India Standard Time': 'MongoDB Session'}
```

.8 USING A COUNTER

Counter is a special column that is changed in increments. For example, we may need a counter column count the number of times a particular book is issued from the library by the student.

Step 1:

```
CREATE TABLE library_book (
        counter_value counter,
        book_name varchar,
        stud_name varchar,
        PRIMARY KEY (book_name, stud_name)
    );
```

Step 2: Load data into the counter column.

```
UPDATE library_book
    SET counter_value = counter_value + 1
        WHERE book_name='Fundamentals of Business Analytics' AND stud_name='jeet';
```

Step 3: Take a look at the counter value.

```
SELECT *
    FROM library_book;
```

Output is:

```
cqlsh:students> select * from library_book;

 book_name                          | stud_name | counter_value
------------------------------------+-----------+---------------
 Fundamentals of Business Analytics |      jeet |             1

(1 rows)
```

Step 4: Let us increase the value of the counter.

```
UPDATE library_book
    SET counter_value = counter_value + 1
        WHERE book_name='Fundamentals of Business Analytics' AND stud_name='shaa
```

Step 4: Let us increase the value of the counter.

```
UPDATE library_book
    SET counter_value = counter_value + 1
        WHERE book_name='Fundamentals of Business Analytics' AND stud_name='shaa
```

Step 6: Update another record for Stud_name "Jeet".

```
UPDATE library_book
    SET counter_value = counter_value + 1
        WHERE book_name='Fundamentals of Business Analytics' AND stud_name='jeet';
```

Step 7: Let us take a look at the counter value, one last time.

```
cqlsh:students> select * from library_book;

 book_name                          | stud_name | counter_value
------------------------------------+-----------+---------------
 Fundamentals of Business Analytics |      jeet |             2
 Fundamentals of Business Analytics |     shaan |             1

(2 rows)
```

7.9 TIME TO LIVE (TTL)

Data in a column, other than a counter column, can have an optional expiration period called TTL (time to live). The client request may specify a TTL value for the data. The TTL is specified in seconds.

```
CREATE TABLE userlogin(
        userid int primary key, password text
);
```

INSERT INTO userlogin (userid, password)

VALUES (1,'infy') USING TTL 30;

```
cqlsh:students> INSERT INTO userlogin (userid, password)
           ... VALUES (1,'infy') USING TTL 30;
```

SELECT TTL (password)

FROM userlogin

WHERE userid=1;

```
cqlsh:students> SELECT TTL (password)
           ... FROM userlogin
           ... WHERE userid=1;

 ttl(password)
---------------
            18

(1 rows)
```

ALTER COMMANDS

1. Create a table "sample" with columns "sample_id" and "sample_name".

```
CREATE TABLE sample(
        sample_id text,
        sample_name text,
        PRIMARY KEY (sample_id)
);
```

```
cqlsh:students> Create table sample (sample_id text, sample_name text, primary key (sample_id));
cqlsh:students>
```

2. Insert a record into the table "sample".

```
INSERT INTO sample(
        sample_id, sample_name)
                VALUES ('S101', 'Big Data');
```

3. View the records of the table "sample".

SELECT *

FROM sample;

```
cqlsh:students> select * from sample;

 sample_id | sample_name
-----------+-------------
      S101 |    Big Data

(1 rows)
```

7.10.1 Alter Table to Change the Data Type of a Column

1. Alter the schema of the table "sample". Change the data type of the column "sample_id" to integer from text.

ALTER TABLE sample

ALTER sample_id TYPE int;

2. After the data type of the column "sample_id" is changed from text to integer, try inserting a record as follows and observe the error message:

```
INSERT INTO sample(sample_id, sample_name)
        VALUES( 'S102', 'Big Data');
```

3. Try inserting a record as given below into the table "sample".

```
INSERT INTO sample(sample_id, sample_name)
        VALUES( 102, 'Big Data' );
```

```
cqlsh:students> Insert into sample(sample_id, sample_name) values( 102, 'Big Data' );
cqlsh:students> select * from sample;

 sample_id  | sample_name
------------+-------------
 13957325 29 |    Big Data
        102 |    Big Data

(2 rows)
```

4. Alter the data type of the "sample_id" column to varchar from integer.

```
ALTER TABLE sample
        ALTER sample_id TYPE varchar;
```

5. Check the records after the data type of "sample_id" has been changed to varchar from integer.

```
cqlsh:students> select * from sample;

 sample_id      | sample_name
----------------+-------------
           S101 |    Big Data
 \x00\x00\x00f  |    Big Data

(2 rows)
```

7.10.2 Alter Table to Delete a Column

1. Drop the column "sample_id" from the table "sample".

```
ALTER TABLE sample
        DROP sample_id;
```

Note: The request to drop the "sample_id" column from the table "sample" does not succeed as it the primary key column.

2. Drop the column "sample_name" from the table "sample".

```
ALTER TABLE sample
        DROP sample_name;
```

0.3 Drop a Table

1. Drop the column family/table "sample".

```
DROP columnfamily sample;
```

0.4 Drop a Database

1. Drop the keyspace "students".

```
DROP keyspace students;
```

11.1 Export to CSV

Objective: Export the contents of the table/column family "elearninglists" present in the "students" database to a CSV file (d:\elearninglists.csv).

Act:

Step 1: Check the records of the table "elearninglists" present in the "students" database.

```
SELECT *
        FROM elearninglists;
```

Step 2: Execute the below command at the cqlsh prompt:

COPY elearninglists (id, course_order, course_id, courseowner, title) TO 'd:\elearninglists.csv';

```
cqlsh:students> copy elearninglists (id, course_order, course_id, courseowner, title) to 'd:\elearninglists
4 rows exported in 0.000 seconds.
cqlsh:students>
```

Step 3: Check the existence of the "elearninglists.csv" file in "D:\". Given below is the content of "d:\elearninglists.csv" file.

	A	B	C	D	E
1	101	1	1001	Subhashini	NoSQL Cassandra
2	101	2	1002	Seema	NoSQL MongoDB
3	101	3	1003	Seema	Hadoop Sqoop
4	101	4	1004	Subhashini	Hadoop Flume

7.11.2 Import from CSV

Objective: To import data from "D:\elearninglists.csv" into the table "elearninglists" present in the "students" database.

Step 1: Check for the table "elearninglists" in the "students" database. If the table is already present truncate the table. This will remove all records from the table but retain the structure of the table our case, the table "elearninglists" is already present in the "students" database. Let us take a look the records of the "elearninglists" before we run the truncate command on it.

```
cqlsh:students> select * from elearninglists;
```

```
cqlsh:students> select * from elearninglists;

 id  | course_order | course_id | courseowner | title
-----+--------------+-----------+-------------+----------------
 101 |            1 |      1001 |  Subhashini | NoSQL Cassandra
 101 |            2 |      1002 |       Seema |   NoSQL MongoDB
 101 |            3 |      1003 |       Seema |    Hadoop Sqoop
 101 |            4 |      1004 |  Subhashini |    Hadoop Flume

(4 rows)
```

Truncate the table using the below command:

TRUNCATE elearninglists;

Note: No record is present in the table "elearninglists". The structure/schema is however preserved. To confirm it by executing the below command at the cqlsh prompt.

```
cqlsh:students> select * from elearninglists;

(0 rows)
```

Step 2: Check for the content of the "D:\elearninglists.csv" file.

	A	B	C	D	E
1	101	1	1001	Subhashini	NoSQL Cassandra
2	101	2	1002	Seema	NoSQL MongoDB
3	101	3	1003	Seema	Hadoop Sqoop
4	101	4	1004	Subhashini	Hadoop Flume

Note: The content in the CSV agrees with the structure of the table "elearninglists" in the "students" database. The structure should be such that the content from the CSV can be housed within it without any issues.

Step 3: Execute the below command to import data from "d:\elearninglists.csv" into the table "elearninglists" in the database "students".

```
COPY elearninglists (id, course_order, course_id, courseowner, title) FROM 'd:\elearninglists.csv';
```

Step 4: Confirm that records have been imported into the table.

```
SELECT *
        FROM elearninglists;
```

Import from STDIN

Objective: To import data into an existing table "persons" present in the "students" database. The data is to be provided by the user using the standard input device.

Step 1: Ensure that the table "persons" exists in the database "students".

```
DESCRIBE TABLE persons;
```

Step 2:

COPY persons (id, fname, lname) FROM STDIN;

```
cqlsh:students> COPY persons (id, fname, lname) FROM STDIN;
[Use \. on a line by itself to end input]
[copy] 1,"Samuel","Jones"
[copy] 2,"Virat","Kumar"
[copy] 3,"Andrew","Simon"
[copy] 4,"Raul","A Simpson"
[copy] \.

4 rows imported in 1 minute and 24.336 seconds.
cqlsh:students>
```

Step 3: Confirm that the records from the standard input device are loaded into the "persons" table existing in the "students" database.

```
SELECT *
        FROM persons;
```

```
cqlsh:students> select * from persons;

 id | fname  | lname
----+--------+-----------
  1 | Samuel |     Jones
  2 |  Virat |     Kumar
  4 |   Raul | A Simpson
  3 | Andrew |     Simon
```

7.11.4 Export to STDOUT

Objective: Export the contents of the table/column family "elearninglists" present in the "student" database to the standard output device (STDOUT).

Step 1: Check the records of the table "elearninglists" present in the "students" database.

```
SELECT *
    FROM elearninglists;
```

```
cqlsh:students> select * from elearninglists;

 id  | course_order | course_id | courseowner | title
-----+--------------+-----------+-------------+-----------------
 101 |            1 |      1001 |  Subhashini | NoSQL Cassandra
 101 |            2 |      1002 |       Seema |   NoSQL MongoDB
 101 |            3 |      1003 |       Seema |    Hadoop Sqoop
 101 |            4 |      1004 |  Subhashini |    Hadoop Flume
```

Step 2: Execute the below command at the cqlsh prompt.

```
COPY elearninglists (id, course_order, course_id, courseowner, title) TO STDOUT;
```

What Cassandra does not support

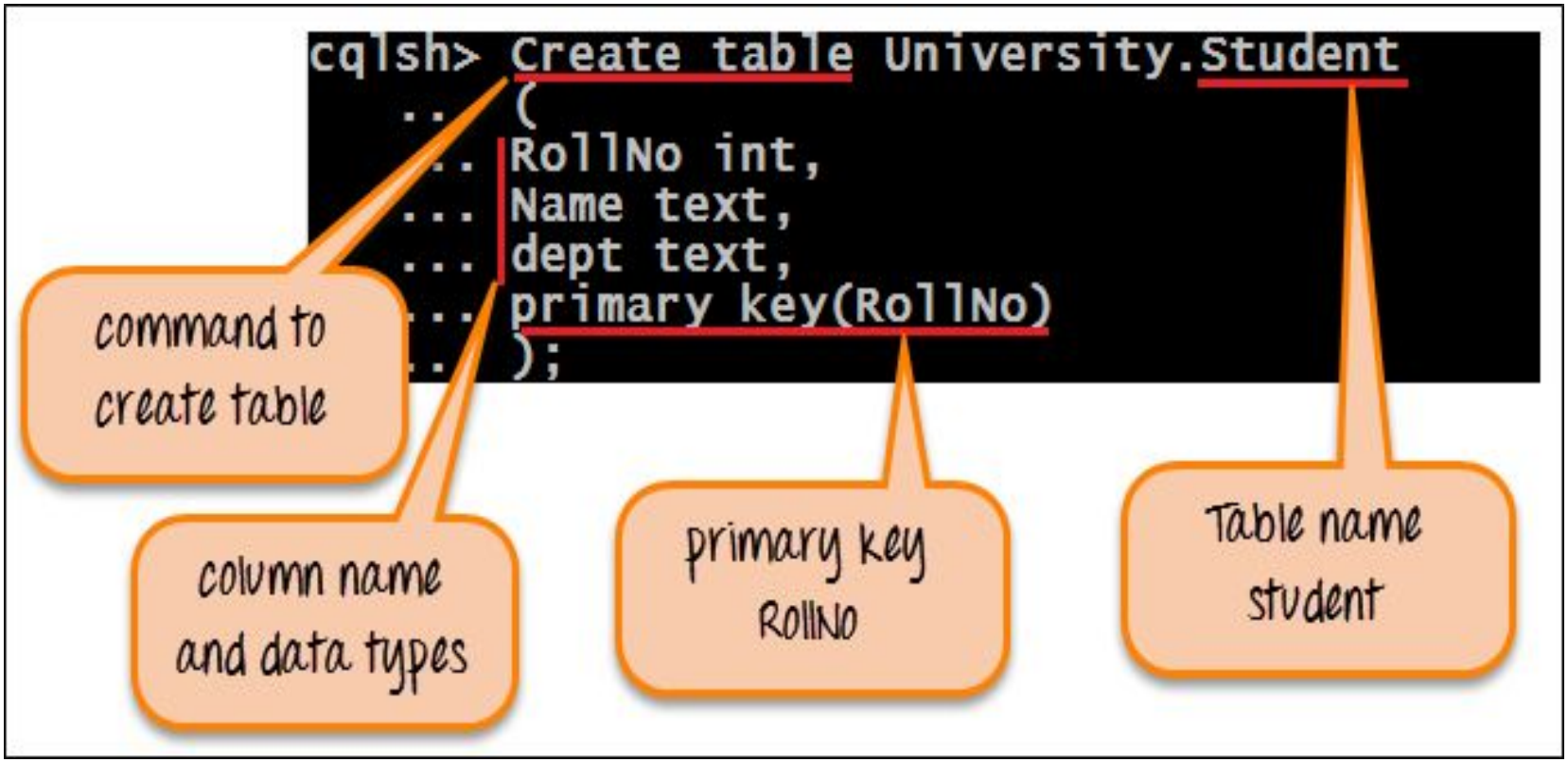
There are following limitations in Cassandra query language (CQL).

1. CQL does not support aggregation queries like max, min, avg
2. CQL does not support group by, having queries.
3. CQL does not support joins.
4. CQL does not support OR queries.
5. CQL does not support wildcard queries.
6. CQL does not support Union, Intersection queries.
7. Table columns cannot be filtered without creating the index.
8. Greater than (>) and less than (<) query is only supported on clustering column.Cassandra query language is not suitable for analytics purposes because it has so many limitations.

Create keyspace University

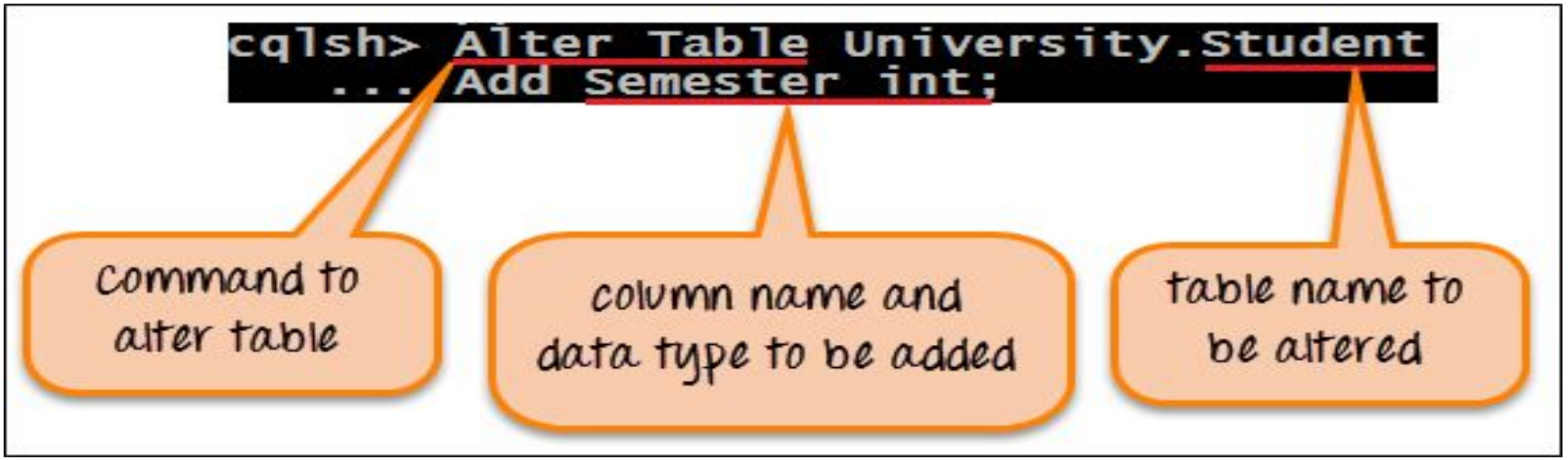
Create keyspace University with replication={'class':SimpleStrategy,'replication_factor' : 3};

Create Student Table Student Rollno, Name, Dept

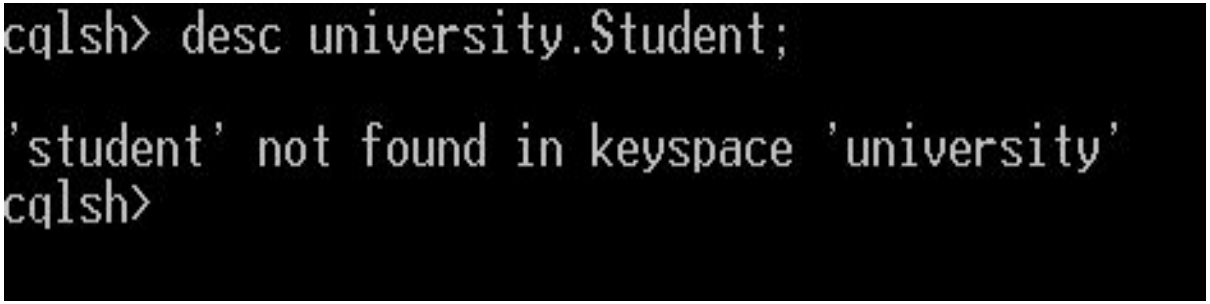


Alter table to add semester

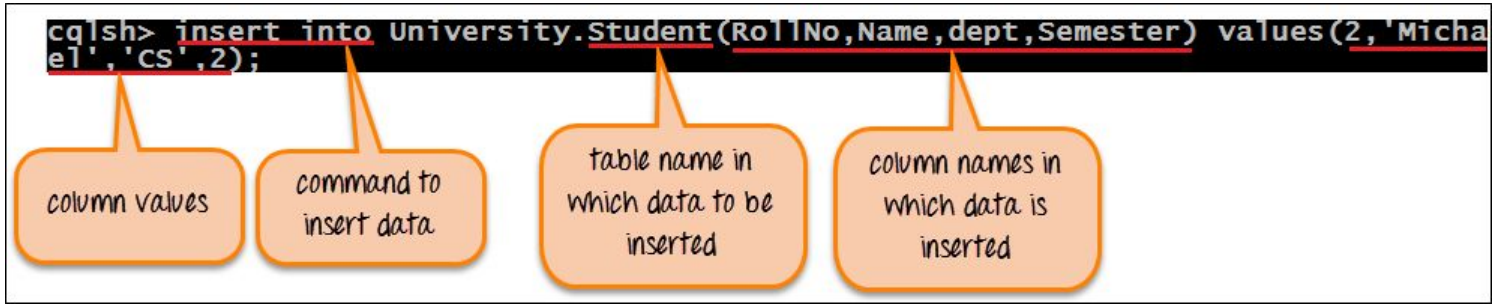
‘Alter Table’ that will add new column in the table Student.



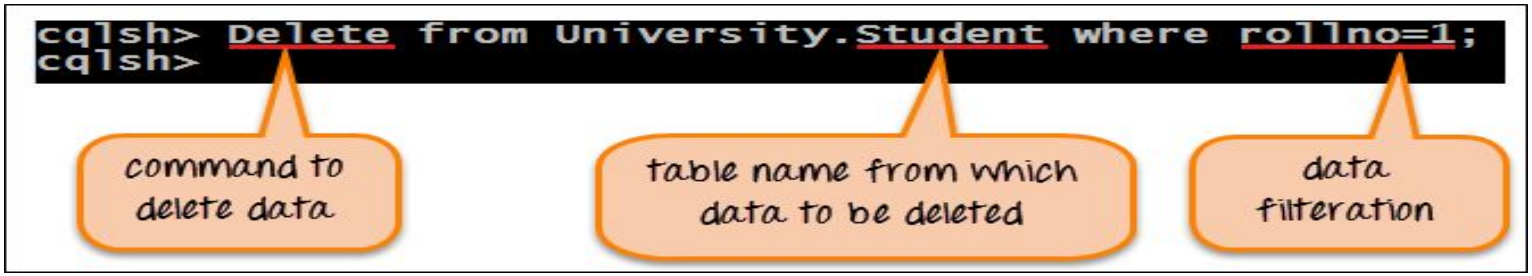
Describe the table contents



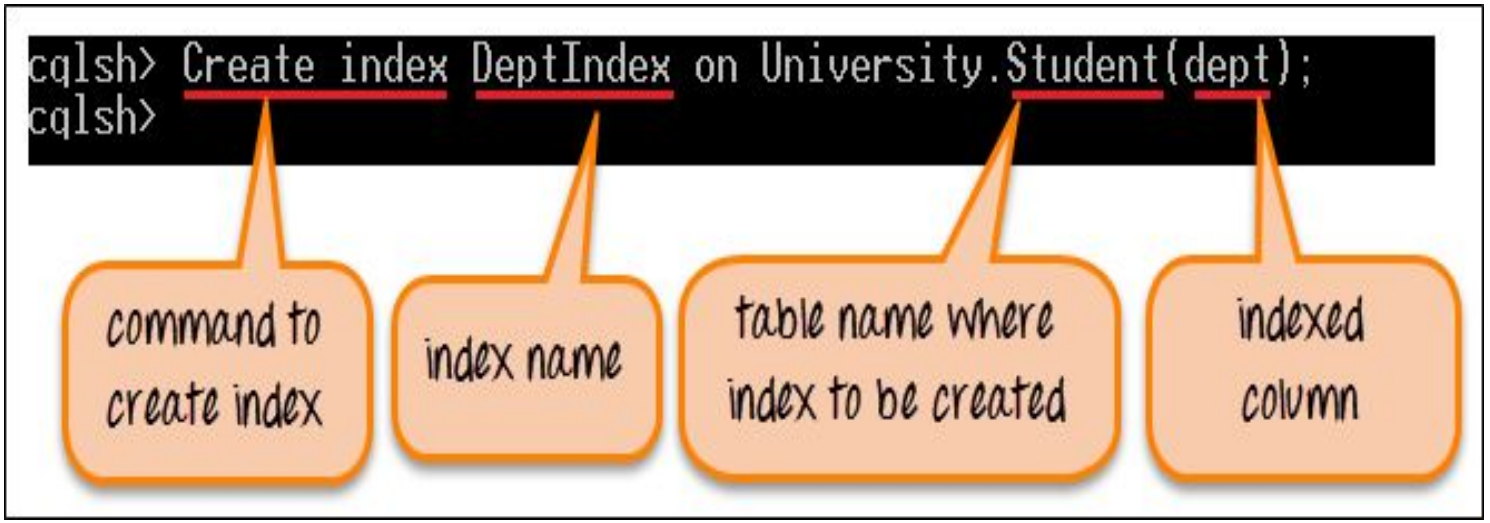
Insert the values into table and update student name of rollno 18



Delete the contents of rollno 1

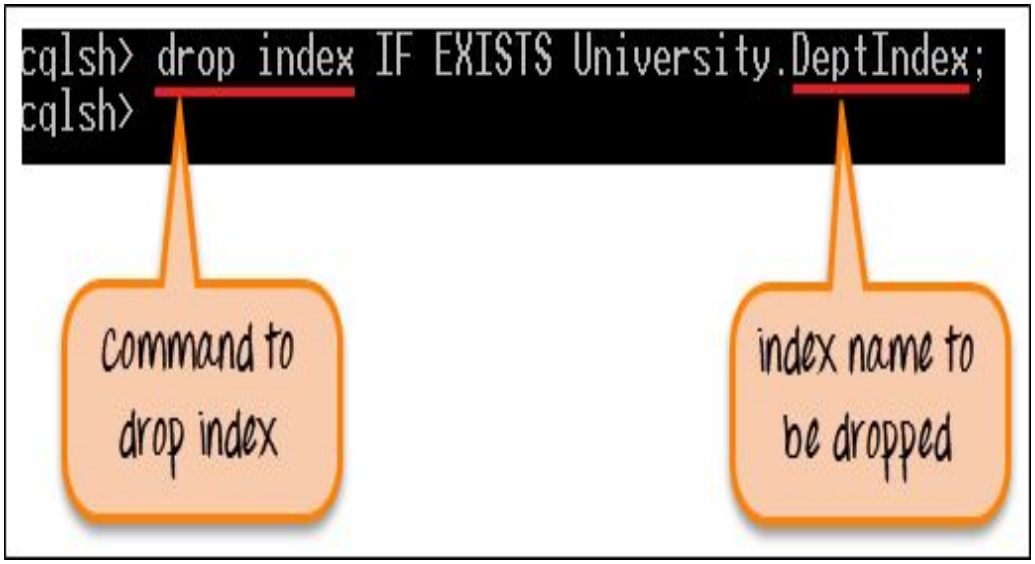


Select all CS dept students

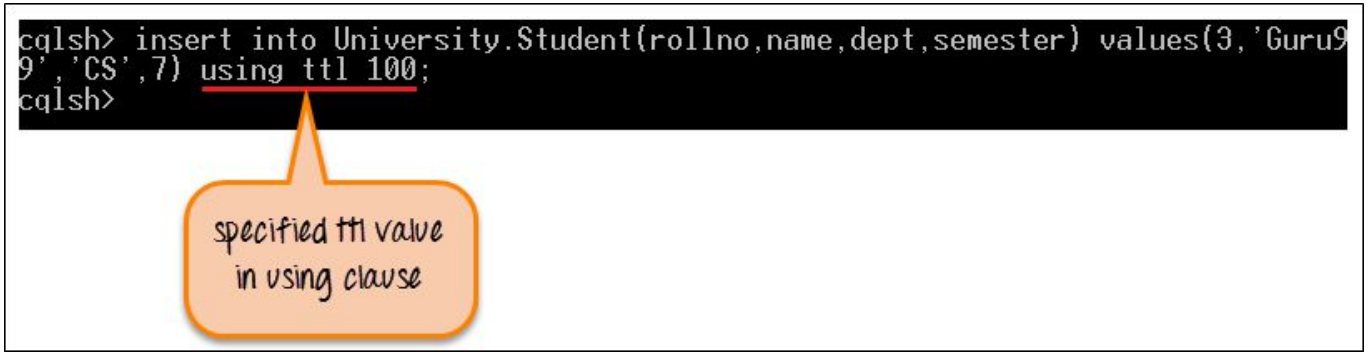



```
cqlsh> select * from University.Student where dept='CS';

 rollno | dept | name    | semester
--------+------+---------+----------
      2 |   CS | Michael |        2

(1 rows)
cqlsh>
```

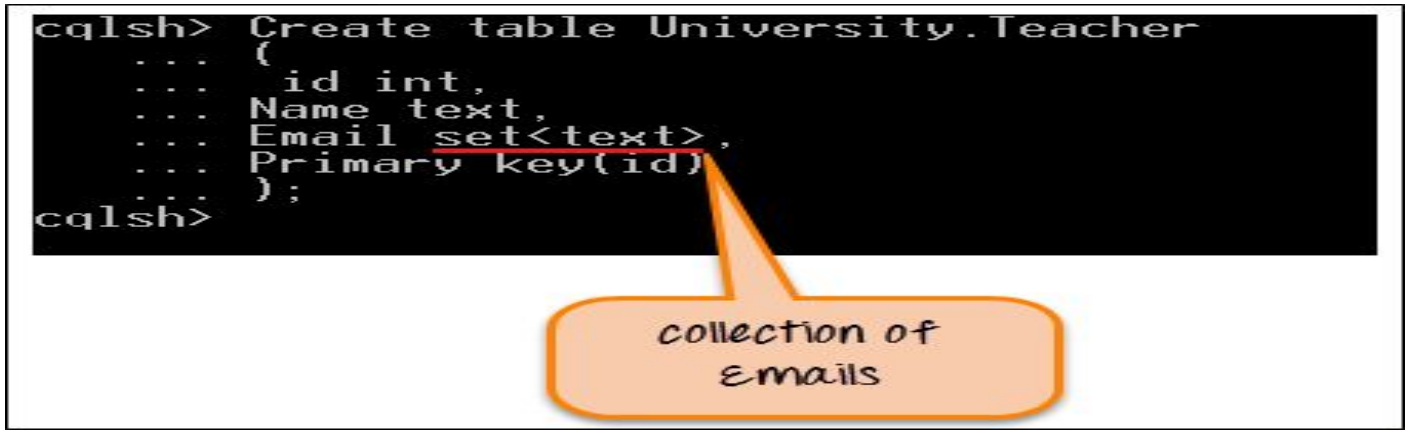
Remove index



Create Teacher Table- Id, Name, Email



Set collection that store multiple email addresses for the teacher.



Add Courses to Teacher table

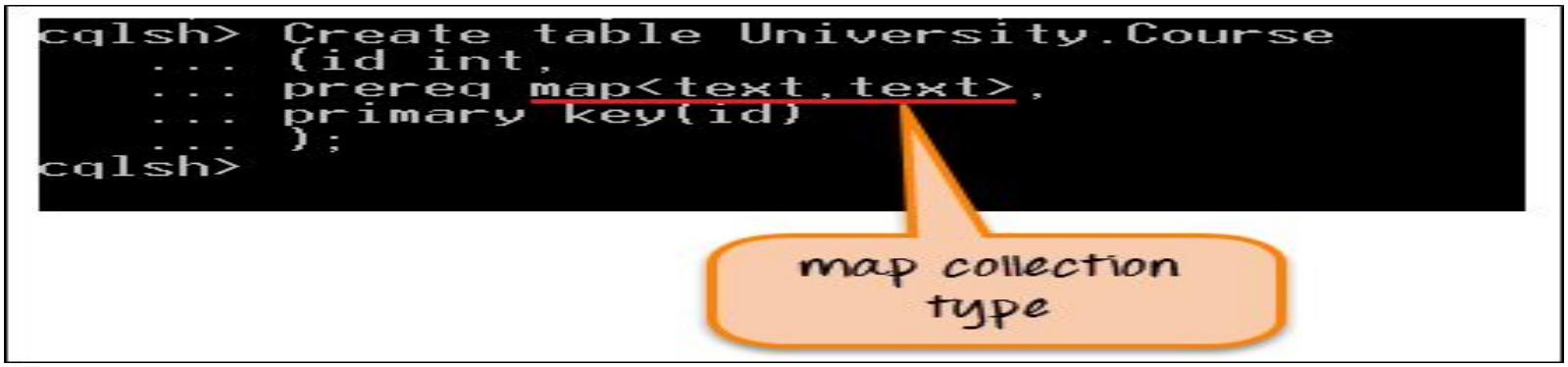
insert in column “coursenames”.

```
cqlsh> insert into University.Teacher(id,Name,Email,coursenames) values(2,'Hamil
ton',{'hamilton@yahoo.com'},['Data Science']);
cqlsh>
```

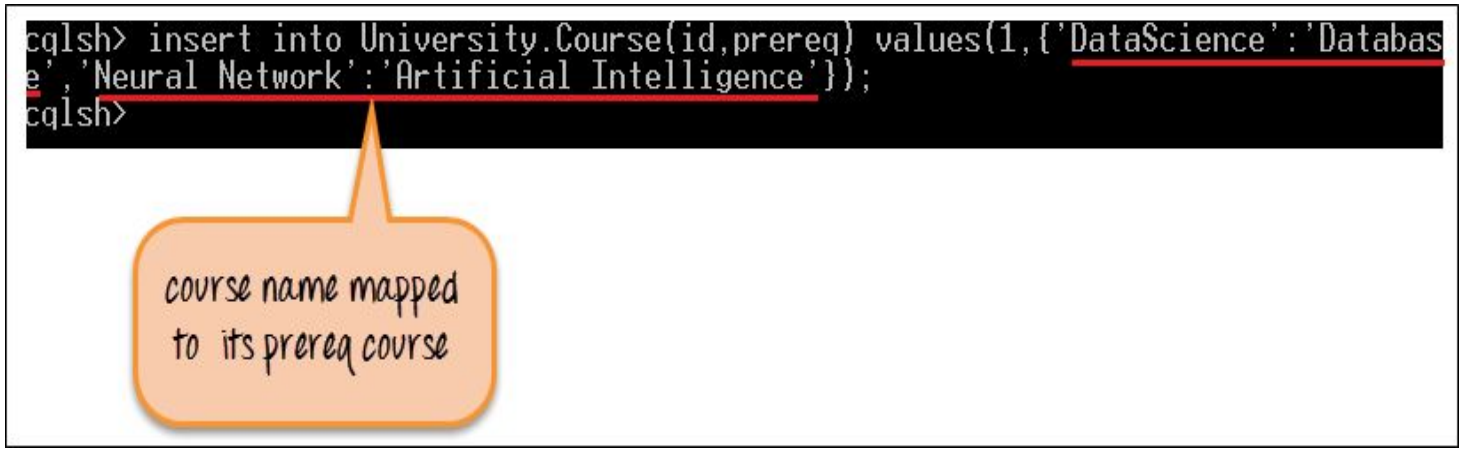
shows the current database state after insertion.

```
 id | coursenames      | email                      | name
----+------------------+----------------------------+----------
  2 | ['Data Science'] | {'hamilton@yahoo.com'} | Hamilton
(1 rows)
```

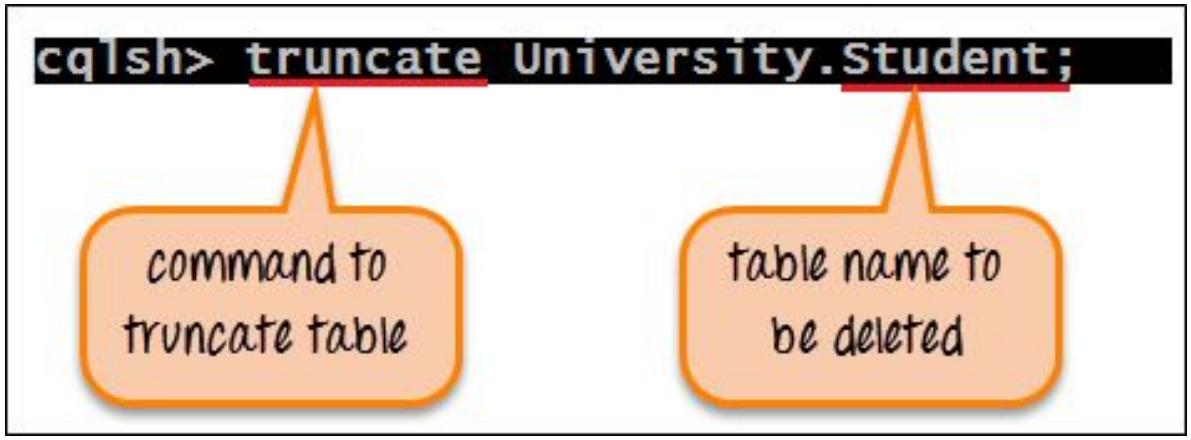
Write query when you want to save course name with its prerequisite course name



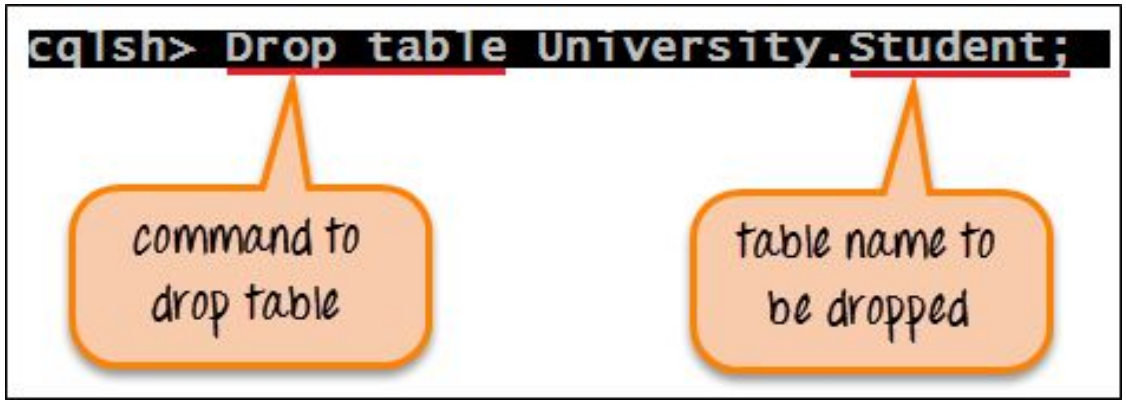
data is being inserted in map collection type



Remove the contents of table



Drop table and Keyspace



Drop keyspace University;